МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

«Системный анализ данных СППР»

Студент группы:ИКБО-42-23 ___Голев С.С.___
(Ф. И.О.студента)

Преподаватель __Железняк Л.М.__
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Метод отжига является одним из эффективных эвристических подходов для решения задач оптимизации. Его основная идея заключается в имитации процесса физического отжига, при котором система постепенно охлаждается и стремится к состоянию с минимальной энергией. В работе рассматривается применение метода отжига Коши для нахождения минимума функции и решения задачи коммивояжёра. Такой подход позволяет находить приближенные оптимальные решения в задачах с большим числом локальных минимумов. Принцип Коши-модификации заключается в особом распределении вероятностей, обеспечивающем более широкий поиск. Таким образом, использование отжига Коши способствует повышению эффективности глобального поиска и снижению вероятности застревания в локальных минимумах.

2 МЕТОД ОТЖИГА

2.1 Цель и задачи практической работы

Целью практической работы является освоение метода отжига Коши и его применение для решения задач оптимизации различного типа. В рамках работы необходимо изучить принципы работы алгоритма, особенности распределения Коши и их влияние на процесс поиска минимума.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Реализовать алгоритм отжига Коши для нахождения минимума заданной функции;
2. Применить метод отжига Коши для решения задачи коммивояжёра;
3. Проанализировать влияние параметров алгоритма на качество и скорость сходимости;
4. Сравнить полученные результаты с классическим методом отжига;
5. Сделать выводы о применимости метода отжига Коши для задач непрерывной и комбинаторной оптимизации.

2.2 Постановка задачи

В рамках практической работы необходимо реализовать нахождение минимума функции с помощью метода Отжиг Коши, также метода Отжига необходимо реализовать решение задачи Коммивояжера.

Как тестовая функция была выбрана функция Била.

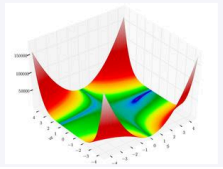	$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$	$f(3, 0.5) = 0$

Рисунок 1 – Функция Била

2.3 Ручной расчёт

Выполним ручной расчёт одной итерации, для функции Била диапазон значений для *x* и *y* равен [–4.5; 4.5].

Возьмем начальную точку: $x_b = 1.0$, $y_b = 1.0$;

Тогда значение функции *f(1,1) = 14.203125.*

Возьмём новую точку со случайными значениями.

Координаты новой точки: $x_i = 2.0$, $y_i = 0.5$;

Тогда значение функции *f(2,0,5) = 1.578125.*

Теперь необходимо сравнить значения, так как *f(1,1) > f(2,0,5),* то к *b* значениям *x* и *y* мы присваиваем у значения, после чего меняем температуру по формуле Коши, то есть делим текущую температуру на номер итерации.

Если бы, оказалось что *f(1,1) < f(2,0,5),* то *b* значения не меняются и происходит переход к новой точке.

2.4 Результат работы метода отжига

Реализуем нахождение минимума функции с помощью функции отжига, для примера реализованы два вида отжига, выполним реализацию на языке Python. Реализация представлена в приложении Б.

```
(venv) PS C:\Users\semen\Desktop\MIREA\System_data_analysis\Practice2> py .\otzhig.py
Эталон: f(3,0.5) = 0
Отжиг
3.935875915185397 0.6474627529874599
0.0727892065728463
Отжиг Коши
2.7181819354219057 0.41515581389134404
0.018313150368945187
```

Рисунок 2 – Пример нахождение минимума методом отжига

2.5 Решение задачи Коммивояжера

Реализуем решение задачи Коммивояжера с помощью метода отжига, выполним реализацию на языке Python. Реализация представлена в приложении Б1.

```
(venv) PS C:\Users\semen\Desktop\MIREA\System_data_analysis\Practice2> py .\gamGraph.py
1 -> 6 -> 3 -> 2 -> 5 -> 4 -> 1
Вес: 22
1 -> 5 -> 6 -> 3 -> 4 -> 2 -> 1
Вес: 15
1 -> 2 -> 6 -> 4 -> 5 -> 3 -> 1
Вес: 15
1 -> 3 -> 6 -> 4 -> 2 -> 5 -> 1
Вес: 16
1 -> 2 -> 3 -> 6 -> 5 -> 4 -> 1
Вес: 21
1 -> 6 -> 2 -> 3 -> 4 -> 5 -> 1
Вес: 17
1 -> 4 -> 2 -> 3 -> 6 -> 5 -> 1
Вес: 16
1 -> 5 -> 6 -> 2 -> 4 -> 3 -> 1
Вес: 14
1 -> 3 -> 5 -> 2 -> 4 -> 6 -> 1
Вес: 14
1 -> 6 -> 4 -> 5 -> 3 -> 2 -> 1
Вес: 11
1 -> 2 -> 6 -> 5 -> 3 -> 4 -> 1
Вес: 14
1 -> 2 -> 4 -> 5 -> 6 -> 3 -> 1
Вес: 17
1 -> 6 -> 4 -> 3 -> 2 -> 5 -> 1
Вес: 20
1 -> 2 -> 3 -> 4 -> 6 -> 5 -> 1
Вес: 19

Кратчайший путь:
1 -> 6 -> 4 -> 5 -> 3 -> 2 -> 1
Вес: 11
```

Рисунок 3 – Пример решения задачи Коммивояжера

2.6 Результат работы нахождения минимума

В ходе практической работы был выполнен ручной расчёт одной итерации метода отжига.

- Начальная точка имела координаты $(x_b,y_b) = (1.0, 1.0)$, значение функции в этой точке составило 14.203125.
- Сгенерирована новая точка $(x_i,y_i)=(2.0, 0.5)$, значение функции в ней оказалось 1.578125.
- Так как новая точка лучше исходной, она была принята, и координаты обновились на $(x_b,y_b)=(2.0, 0.5)$.

Данный расчёт наглядно показывает принцип работы метода: переход к новой точке осуществляется, если значение функции уменьшается, при этом возможен переход к худшей точке с определенной вероятностью.

Далее была выполнена кодовая реализация метода отжига и метода отжига Коши, которая позволила:

1. Автоматически находить минимум функции.
2. Решать задачу коммивояжера, минимизируя суммарное расстояние маршрута.
3. Сравнивать эффективность обычного отжига и отжига Коши, показывая, что модификация Коши обеспечивает более широкий поиск и снижает риск застревания в локальных минимумах.

В результате работы показано, что методы отжига являются эффективными инструментами как для задач непрерывной, так и для комбинаторной оптимизации.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были реализованы алгоритмы нахождения минимума функции и решения задачи коммивояжёра методом отжига Коши. Проведённые эксперименты показали, что данный метод способен успешно находить решения, близкие к оптимальным, даже для сложных задач. Благодаря особенностям распределения Коши достигается баланс между исследованием и уточнением решений. Метод отжига продемонстрировал устойчивость и адаптивность к различным типам функций и графов. Полученные результаты подтверждают эффективность стохастических подходов для задач оптимизации. В целом, использование отжига Коши позволяет повысить точность и надёжность решений в задачах комбинаторной и непрерывной оптимизации.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).

2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.

3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.

4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

ПРИЛОЖЕНИЯ

Приложение Б – Код программы “Метод отжига”

Приложение Б – Код программы “Решение задачи Коммивояжера”

Приложение А

Код программы Онтологии

Листинг Б.1 — Основной алгоритм программы

```
import numpy as np

def f(x, y):
    return (1.5 - x + x*y)**2 + (2.25 - x + x*(y**2))**2 + (2.625 -
x + x*(y**3))**2

def otzhig(T = 1,Tmin = 1e-100, alpha = np.random.uniform(0.1, 1)):
    xb = np.random.uniform(-4.5, 4.5)
    yb = np.random.uniform(-4.5, 4.5)

    while (T > Tmin):
        xi = np.random.uniform(-4.5, 4.5)
        yi = np.random.uniform(-4.5, 4.5)

        funci = f(xi,yi)
        funcb = f(xb,yb)

        if (funci - funcb <= 0):
            xb = xi
            yb = yi
        else:
            if (np.exp(-(funci - funcb) / T) > np.random.uniform(0,
1)):
                xb = xi
                yb = yi

        T *= alpha

    return xb, yb
def otzhigKoshi(T = 1,Tmin = 1e-1000, k=1):
    xb = np.random.uniform(-4.5, 4.5)
    yb = np.random.uniform(-4.5, 4.5)

    while (T > Tmin):
        xi = np.random.uniform(-4.5, 4.5)
        yi = np.random.uniform(-4.5, 4.5)

        funci = f(xi,yi)
        funcb = f(xb,yb)

        if (funci - funcb <= 0):
            xb = xi
            yb = yi
```

Листинг Б.2 — Продолжение листинга Б.1

```
else:
            if (np.exp(-(funci - funcb) / T) >
np.random.uniform(0, 1)):
                xb = xi
                yb = yi

        T /= k
        k+=1

    return xb, yb

print("Эталон: f(3,0.5) = 0")
xb, yb = otzhig()
print("Отжиг")
print(xb, yb)
print(f(xb, yb))

print("Отжиг Коши")
xb, yb = otzhigKoshi()
print(xb, yb)
print(f(xb, yb))
```

Листинг Б.1.1 — Решение задачи Коммивояжера

```
import numpy as np

def createGraph(cNodes):
    structure = {}
    step = 1
    for i in range (cNodes):
        structure[str(i + step)] = []

    for i in range (cNodes):
        for j in range (i + 1,cNodes):
            w = np.random.randint(1, 6)

            structure[str(i + step)].append((str(j + step), w))
            structure[str(j + step)].append((str(i + step), w))

    return structure, step

def findWay(cNodes, struct, step, T = 1,Tmin = 1e-1000, k = 1):
    start = str(step)
    curr = start
    next_step = ''

    pb = [start]
    wb = 0

    for i in range (cNodes - 1):
        while True:
            next_step = str(np.random.randint(step, cNodes + 1))
            if next_step not in pb:
                break
        pb.append(next_step)

        for l in struct[curr]:
            if l[0] == next_step:
                wb += l[1]

        curr = next_step
    pb.append(start)
    for l in struct[curr]:
            if l[0] == start:
                wb += l[1]

    print(' -> '.join(pb))
    print('Вес:', wb)

    while T > Tmin:
        pi = [start]
        wi = 0
```

Листинг Б.1.2 — Продолжение листинга Б.1.1

```
        for i in range (cNodes - 1):
            while True:
                next_step = str(np.random.randint(step, cNodes +
1))
                if next_step not in pi:
                    break
            pi.append(next_step)

            for l in struct[curr]:
                if l[0] == next_step:
                    wi += l[1]

            curr = next_step
        pi.append(start)
        for l in struct[curr]:
                if l[0] == start:
                    wi += l[1]

        if (wi - wb <= 0):
            wb = wi
            pb = pi
        else:
            if (np.exp(-(wi - wb) / T) > np.random.uniform(0,
1)):
                wb = wi
                pb = pi

        print(' -> '.join(pi))
        print('Вес:', wi)
        T /= k
        k += 1
    return pb, wb


cNodes = 6
structure, step = createGraph(cNodes)
# print(structure)

path, weight = findWay(cNodes, structure, step)
print('\nКратчайший путь:')
print(' -> '.join(path))
print('Вес:', weight)
```